CoSS: Proposing a Contract-Based Storage System for HPC

Matthieu Dorier, Matthieu Dreher, Tom Peterka, Robert Ross

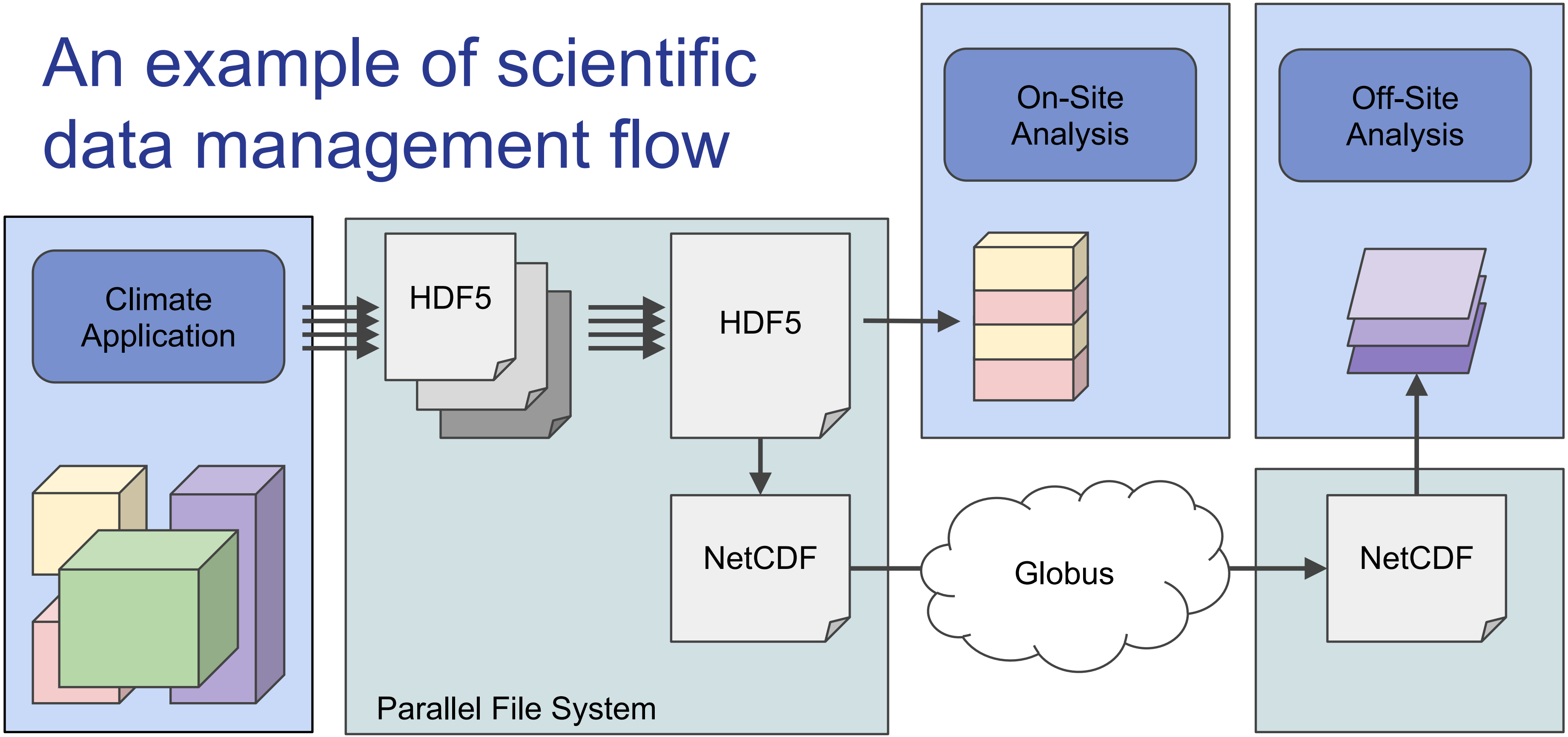
Argonne NATIONAL LABORATORY

PDSW-DISCS Workshop
November 13, 2017

HPC data management is centered around files

Parallel file systems kill scientific productivity

An example of scientific data management flow

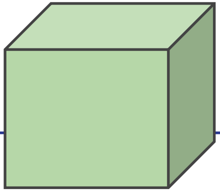


Metadata is all over the place

Metadata is all over the place

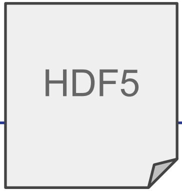
Data Model

Variable names, type, dimensions, description, unit, relation to other variables, organization in groups, etc.



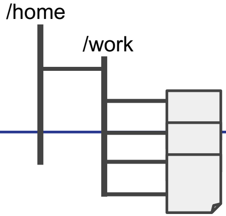
Data Format

Mapping between data model and underlying file, data layout (chunking, compression, etc.), headers, footers, etc.



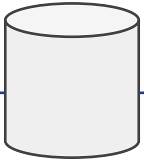
File Metadata

File name, directory, owner, permissions, creation time, modification time, extended attributes (xattr), etc.



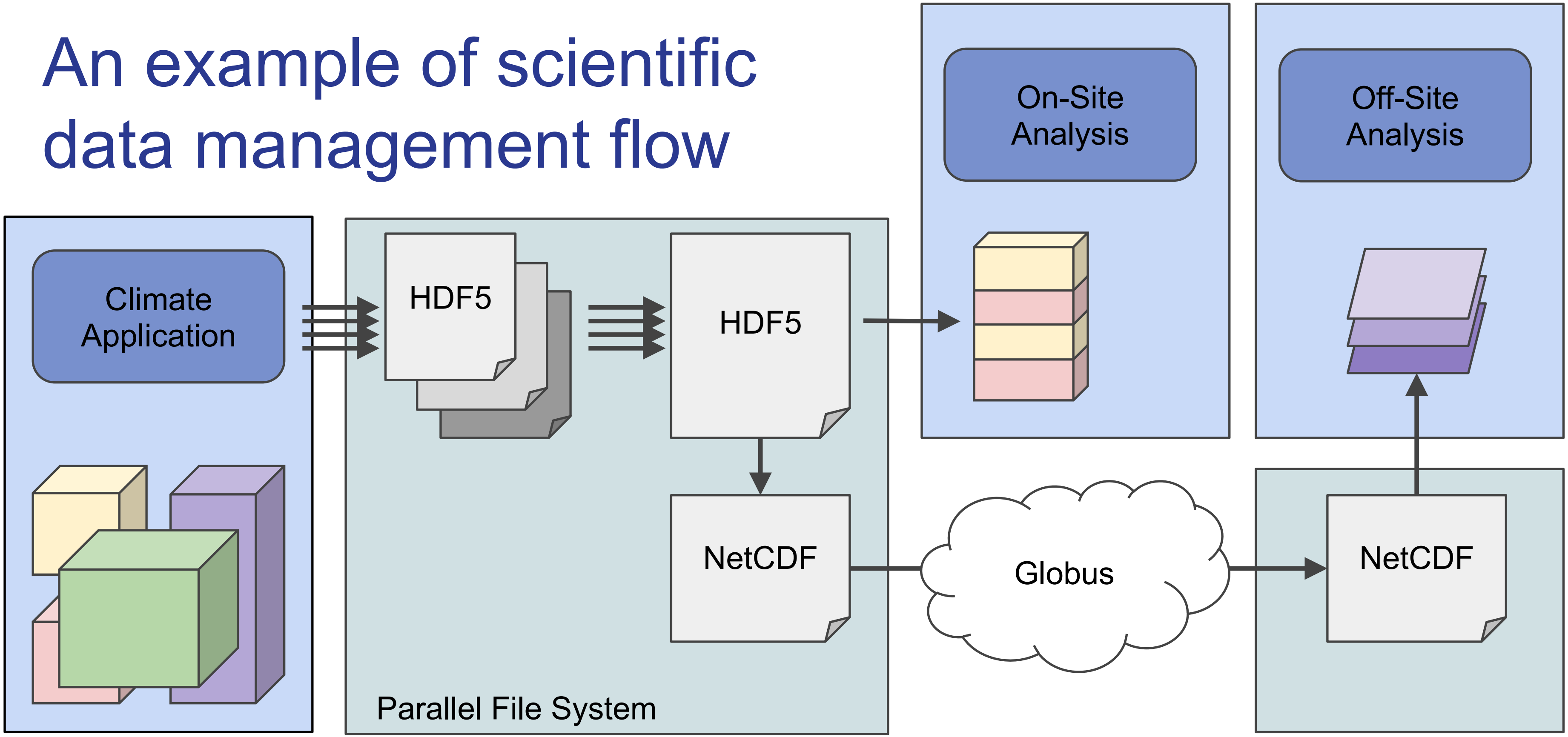
Distribution

Mapping from a file to a set of stripes in storage targets, replication, erasure coding, etc.



Current storage systems assume
what is produced = what will be consumed

An example of scientific data management flow



Let’s summarize the problems

Problems with the file-centric approach

- Parallel file systems kill scientific productivity
  - Need to spend time optimizing I/O on new platforms
  - Need to develop multiple backends for multiple data formats
  - Need to write tools to convert, extract, process data
- Metadata is all over the place
  - Data formats needed to add semantics ⇒ adds software complexity
  - File systems do not know about the semantics of the data
  - File systems cannot optimize storage according to semantics
- Storage assumes what is written is what will be read
  - Storage cannot transform data to optimize future accesses
  - Forces users to create redundancy

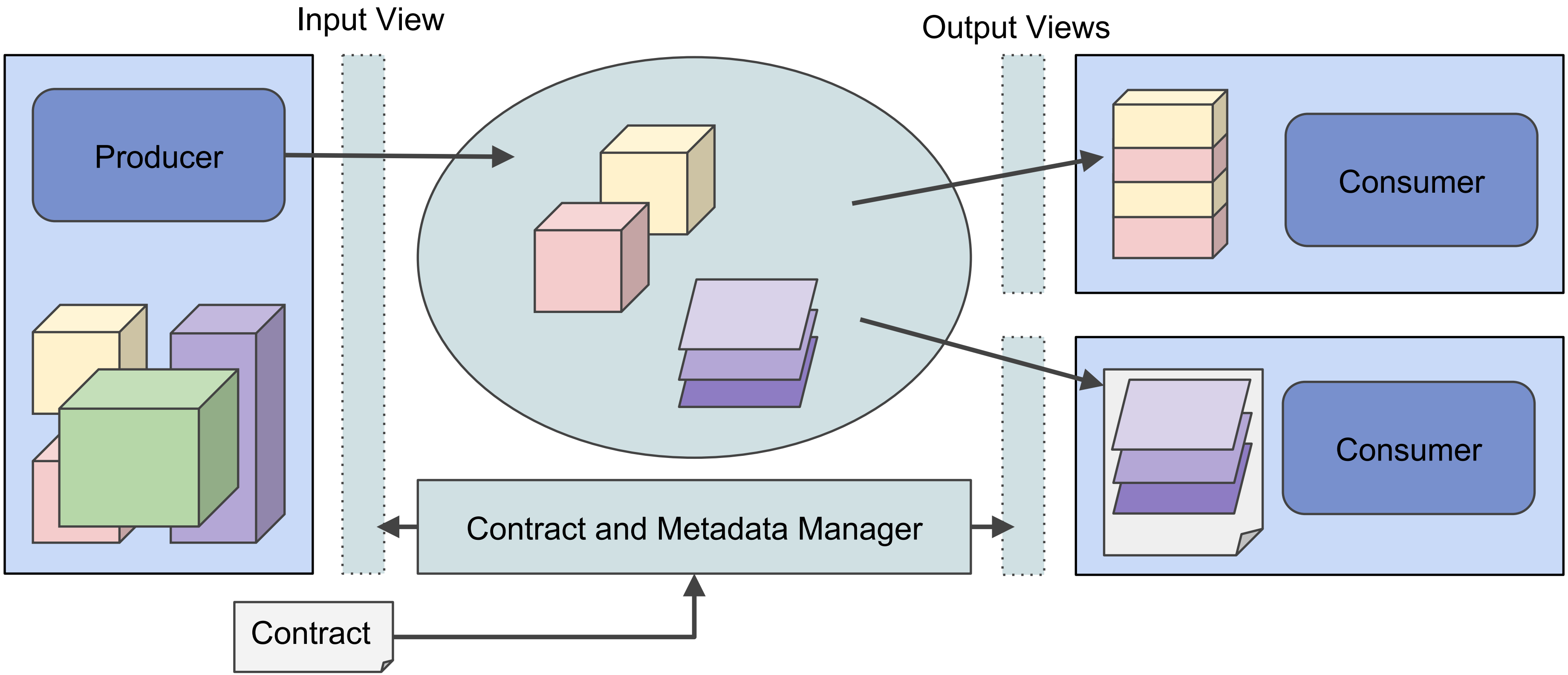
COSS

COntract-base Storage System

#1 - Data objects and the data models that describe them are the key concepts to (HPC) data management.

#2 - The user knows what is going to be produced, what must be retained, and how that data will be later consumed.

Overview of CoSS



CoSS’ Contracts

- Data model: describing the data as much as we can
  - Names, unit, description, etc. of objects
  - Relationship between objects
  - Builds “virtual” objects from other objects, e.g. a mesh from its coordinate objects
  - Similar to HDF5 metadata + an XDMF file, an ADIOS XML file, or a Damaris XML file
- Views: placing constraints on what CoSS can do with the data
  - Describe how the objects will be written to storage (input view)
  - Describe how the objects will be read from storage (output view)
  - Views must be matching
  - Express permissions

More on views: examples

Input View

Defines what will be written by the application

- Variables
  - type, dimensions
  - layout in memory
  - access (single writer, multiple writers)

Example

“temperature” is a 3D array of double-precision values, with dimensions 128x128x16, in row-major memory layout, written by blocks by multiple processes

Output View

Places constraints on the storage system, defining how it is allowed to handle the stored data in order to satisfy future usage

Examples

“temperature” will be accessed as a 2D slice at level z=0; as single-precision

“temperature” should be exposed within an HDF5 to enable legacy code to read it

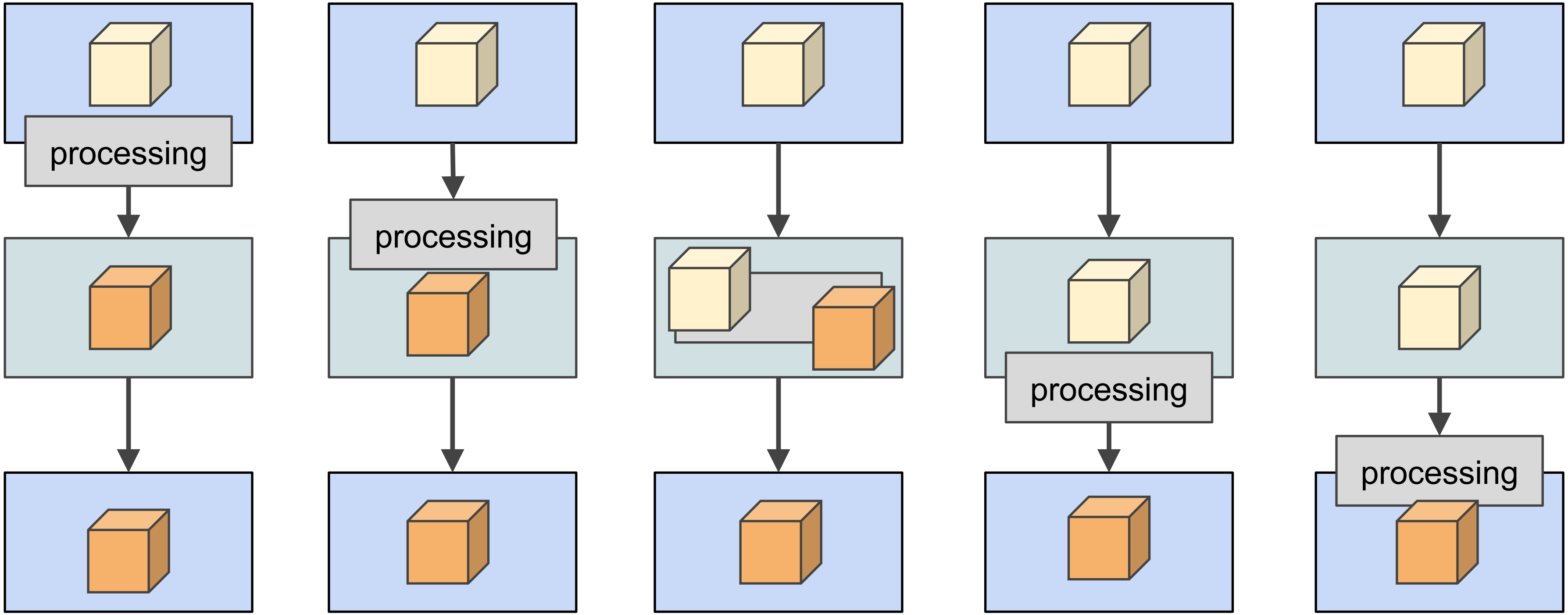
What can CoSS do with such knowledge?

- Store the objects in the form that is
  - The most likely to be accessed
    - ex: reorganizing object layout
  - The most generic (in terms of possible transformations)
    - ex: keep data as written by producer
  - The most consistent with the format of other related objects
    - ex: apply the same transformation to the coordinates of a same mesh
  - The most space-efficient
    - ex: applying compression, downsampling

What can CoSS do with such knowledge?

- Decide when, where, and how to apply some transformations
  - CoSS can transform the data on the client
  - CoSS can do the transformation on the storage side
  - CoSS can launch a job by itself to perform transformations
    - Requires interactions with the job manager
  - CoSS can transform on the consumer side

CoSS decides when and where to process data



A few words on CoSS' object store

- Similar to traditional object stores (e.g. RADOS)
- Metadata manager gives ALL the semantics to the objects
  - High-level semantics as in HDF5, NetCDF, etc.
  - Relationship between objects, as in XDMF, Damaris, etc.
  - Permissions, access policies, as in a parallel file system
- Accesses can be
  - **Atomic**: full object accessed once by one process
  - **Chunked**: full object accessed by chunks from multiple processes
  - **Log-based**: processes append entries until object is closed

Organizational model

- Project
  - Equivalent of a directory in which all the data related to a set of executions are gathered
  - Has a name, creation date, permissions
  - Contains a contract providing data models and constraints on the data
  - Contains branches
- Branche
  - Equivalent of a subdirectory containing the data produced by a single execution
  - Has a creation date and a closing date
  - Contains a set of epochs
- Epoch
  - Consistent set of objects produced by the application
  - Correspond to an iteration of output in a BSP application

Renegotiating contracts

Renegotiating a contract on an existing project will make CoSS try to make the existing data satisfy the new contract.

Restricting a contract

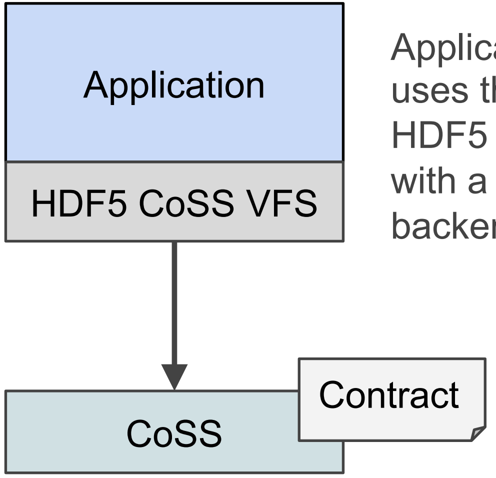
Me: “I won’t need the *temperature* field anymore”
CoSS: “Thanks for letting me know, I needed space, I’ll erase your previous temperature objects”
Me: “From now on, single-precision is enough for the pressure field”
CoSS: “Thanks, I’m lazy and won’t change what you already wrote, but I’ll take that into account”

Widening a contract

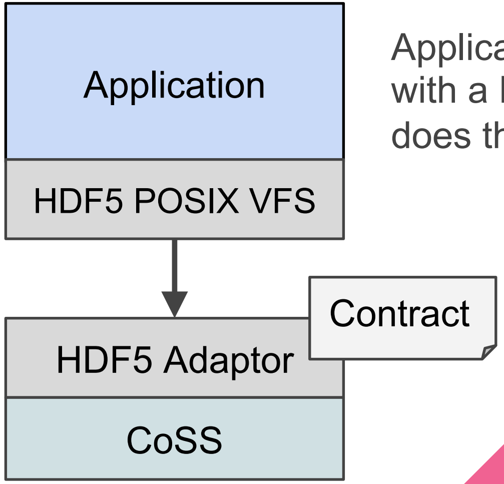
Me: “I’ll need an HDF5 file view from my data”
CoSS: “Sure, here you go”
Me: “From now on, don’t lossy-compress the temperature field, I need the raw data”
CoSS: “You’re in luck, I was lazy and hadn’t compressed it to begin with”
Me: “Give me the temperature field that I initially told you to discard”
CoSS: “Sorry, can’t do that, I did discard it”

Adapting legacy codes

- Many codes moving to in situ analysis could easily move to CoSS
- High-level data libraries and middleware (HDF5, NetCDF, Damaris, ADIOS, etc.) could have a CoSS-enabled backend



Application uses the HDF5 API with a CoSS backend



Application uses the HDF5 API with a POSIX backend, CoSS does the translation in-storage

Building CoSS is easy

Data Model

Inspired from:

Visualization packages: VTK, VisIt, ParaView, etc.

Data formats: HDF5, XDMF, NetCDF, ADIOS BP, etc.

Contracts

Using:

XML (like ADIOS, XDMF, Damaris), or JSON (like Conduit), or YAML, etc.

Programming languages: Python, Lua, Ruby, etc.

Storage System

Based on (or inspired by):

Object store: RADOS

Object-based storage systems used today as backends for PFS

From the Cloud landscape: Swift, etc.

Conclusion

CoSS: a Contract-based Storage System for HPC

- **Idea 1**: object-centric instead of file-centric
- **Idea 2**: high-level semantics available to the storage system
- **Idea 3**: place constraints on how data is produced and consumed

What it enables

- Smart-processing (possibly in-storage)
- Wider range of optimizations possible because of additional knowledge of intended use of the data

Implementation can rely on state-of-the-art storage, I/O, and in-situ techniques

Acknowledgements

This material was based upon work supported by the U.S. Department of Energy, the Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02- 06CH11357, program manager Lucy Nowell.

This work was done in the context of the DOE SSIO project "Mochi" (http://press3.mcs.anl.gov/mochi/), a Software Defined Storage Approach to Exascale Storage Services.